CMPS 312

Declarative UI using Jetpack Compose

Dr. Abdelkarim Erradi

CSE@QU

Outline

1. Jetpack Compose Key Concepts
2. Material Design UI Components
3. Modifiers
4. Layouts
5. State

Jetpack Compose Key Concepts

Back

Declarative UI is a major trend

- Describe WHAT to see NOT HOW

Flutter: Google's UI toolkit for building natively compiled applications for mobile, web and desktop from a single codebase

SwiftUI: Apple’s new declarative framework for creating apps that run on iOS

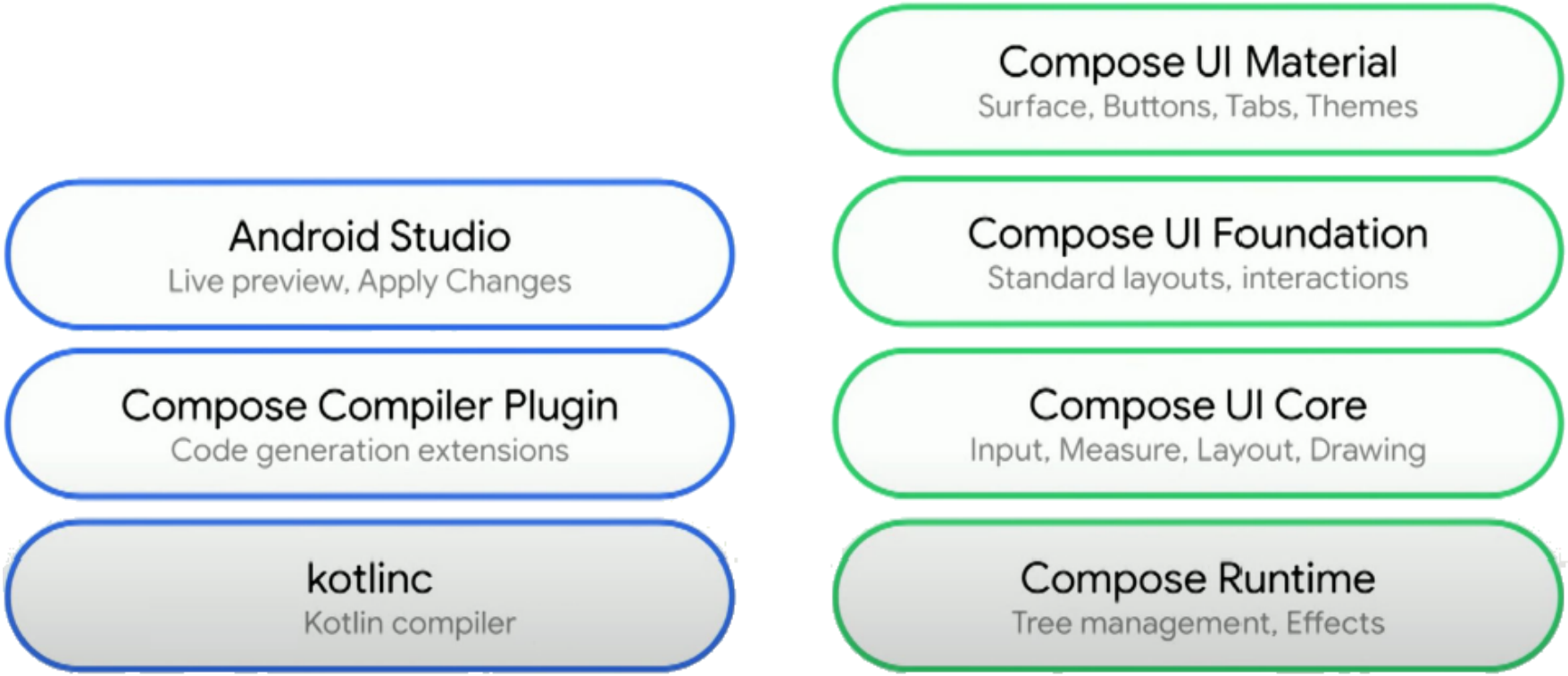
React: A JavaScript library for building user interfaces

Jetpack Compose: a **modern toolkit** for building native Android UI (released July 2021)

Jetpack Compose

- Jetpack Compose is a **modern toolkit** for building native Android UI
  - It simplifies UI development with less code and intuitive Kotlin APIs
- A **declarative component-based programming model** inspired by other declarative UI frameworks such as React and Flutter
  - UI is built using composable functions
    - Each function define a piece the app's UI programmatically by **describing WHAT to see (**layout/ look and feel) **NOT HOW**
  - As state changes the UI automatically updates (Reactive UI)

Inside Jetpack Compose



Build time (development host) **Runtime (on device)**

[What's new in Jetpack Compose (Android Dev Summit '19)]

How to define a piece of UI?

- Composable **function**:
  - Just a function annotated with **@Composable**
  - Take some <u>inputs</u> and return a piece of <u>UI</u>
  - Describe the UI based on the provided parameters
    - Describes **WHAT to see** NOT HOW
    - @Composable does the magic for HOW
  - **UI = f(state) : UI is a visual representation of state**
  - Can call other composable functions or being called in other composable functions

UI as a function

String →
```
fun Greeting(name: String) =
    println("Hello, $name")
```
→ stdout

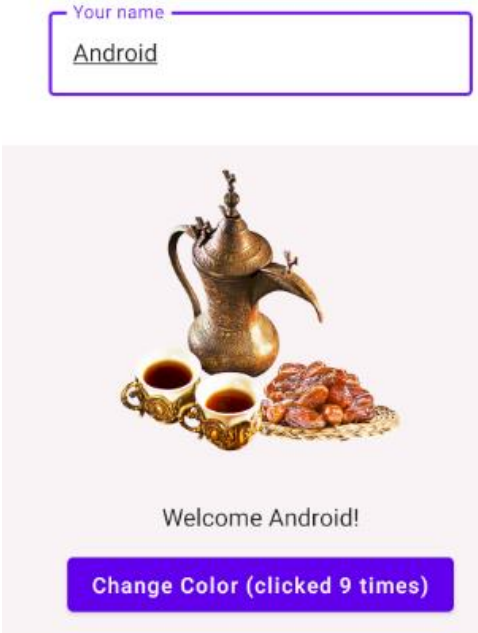
Mark as a composable

Data →
```
@Composable
fun Greeting(name: String) =
    Text("Hello, $name")
```
→ UI

UI as a function (List of Composables)

```
@Composable
fun SurahsList(surahs: List<Surah>) {
    Column(modifier =
       Modifier.verticalScroll(rememberScrollState())
    ) {
        if (surahs.isEmpty()) {
            Text("Loading surahs failed.")
        } else {
            surahs.forEach {
                SurahCard(surah = it)
            }
        }
    }
}
```

Composition



- **UI = Composition of UI functions**
- **UI Function = Building blocks**

```
@Composable
fun WelcomeScreen() {
    var userName by remember { mutableStateOf( value: "Android") }
    Column {  this: ColumnScope
        NameEditor(name = userName, nameChange = { newName -> userName = newName })
        Welcome(userName)
    }
}

@Composable
fun NameEditor(name: String, nameChange: (String) -> Unit) {...}

@Composable
fun Welcome(name: String) {...}
```

Entry point to Compose world

- When the app launches it creates and starts the Main Activity
- **Activity** acts as a container to load the **main UI screen**
  - Using **setContent** in the **onCreate** method

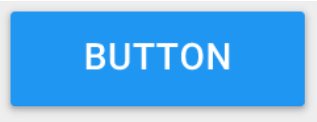
```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            MyAppTheme {
                Surface(color = MaterialTheme.colors.background) {
                    Greeting("Android")
                }
            }
        }
    }
}

@Composable
fun Greeting(name: String) {
    Text(stringResource(R.string.hello, name))
}
```

Material Design UI Components

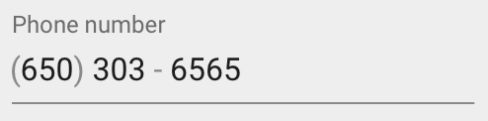
Back

UI Components

Button

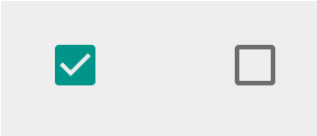


EditText



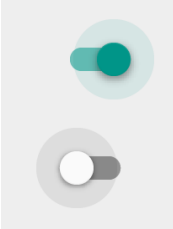
SeekBar

CheckBox



RadioButton

Switch

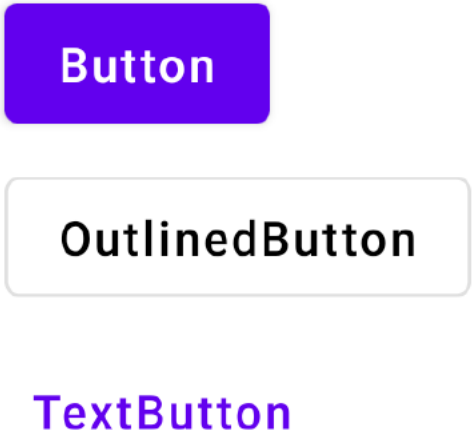


Back

Button


```
Button(
    text = "Button",
    icon: Icon? = myIcon,
    textStyle = TextStyle(...),
    spacingBetweenIconAndText = 4.dp,
    ...
)
```

```
Button(onClick = {}) {
    Text("Button")
}

OutlinedButton(onClick = {}) {
    Text("OutlinedButton")
}

TextButton(onClick = {}) {
    Text("TextButton")
}
```

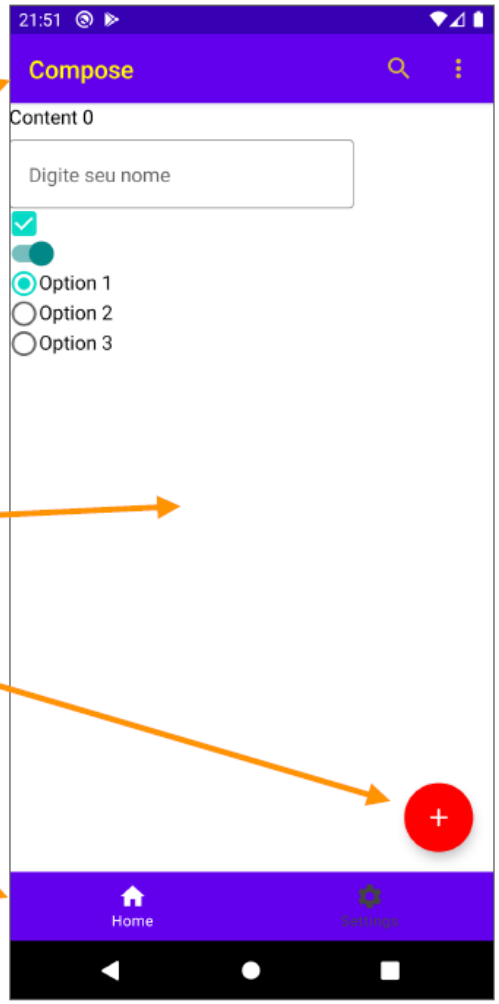


Image

```
Image(painter =
    painterResource(R.drawable.img_compose_logo),
    contentDescription = "Jetpack compose logo",
    modifier = Modifier.sizeIn(maxHeight = 300.dp))
```

Scaffold

```
Scaffold(
    topBar = {...},
    floatingActionButton = {...},
    bottomBar = {...}
) {...}
```

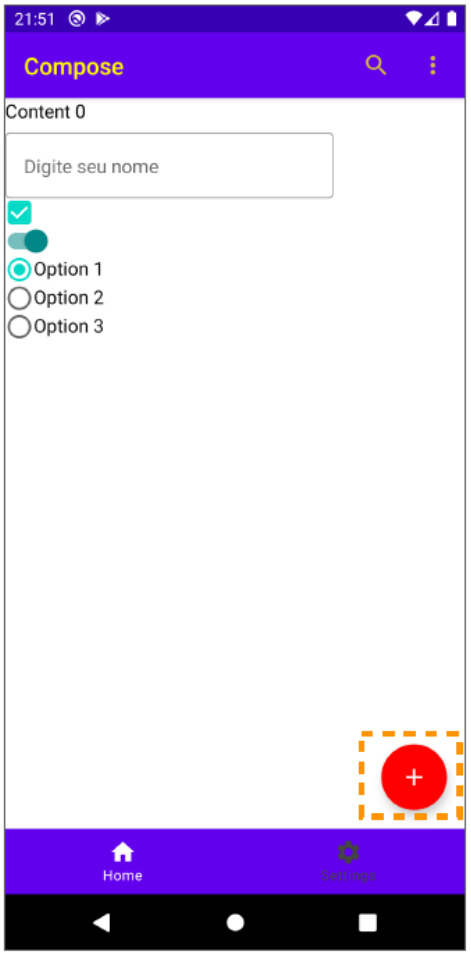


TopAppBar

```
TopAppBar(
    title = { Text(text = "Compose") },
    backgroundColor = MaterialTheme.colors.primary,
    contentColor = Color.Yellow,
    actions = {
        IconButton(onClick = {}) {
            Icon(Icons.Default.Search, "Search")
        }
        IconButton(
            onClick = { … }
        ) {
            Icon(Icons.Filled.MoreVert, "More")
            DropdownMenu(…)
        }
    }
)
```

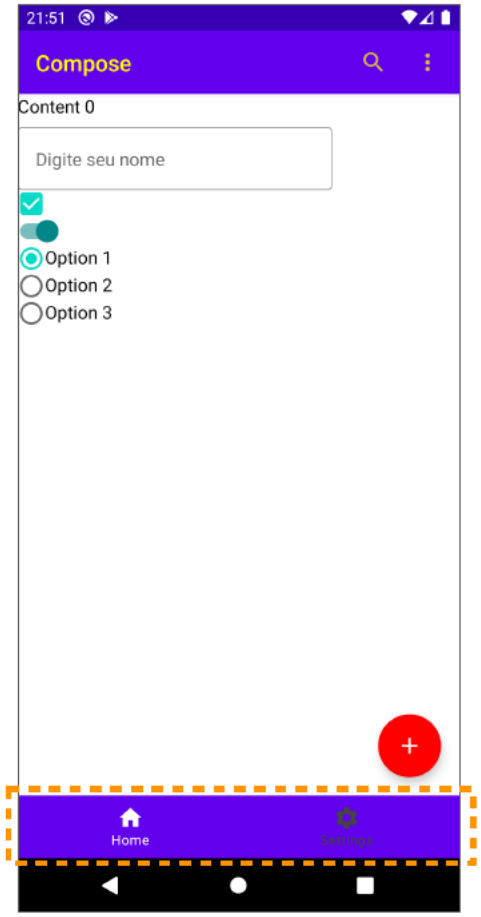
FloatingActionButton

```
FloatingActionButton(
    onClick = { … },
    backgroundColor = Color.Red,
    contentColor = Color.White
) {
    Icon(Icons.Filled.Add, "Add")
}
```



BottomAppBar

```
BottomAppBar(
    backgroundColor = MaterialTheme.colors.primary,
    content = {
        BottomNavigationItem(
            icon = { Icon(Icons.Filled.Home) },
            selected = selectedTab == 0,
            onClick = { selectedTab = 0 },
            selectedContentColor = Color.White,
            unselectedContentColor = Color.DarkGray,
            label = { Text(text = "Home") }
        )
        BottomNavigationItem(…)
    }
)
```



Material Design Components

- Using MDC to make your app look great easily

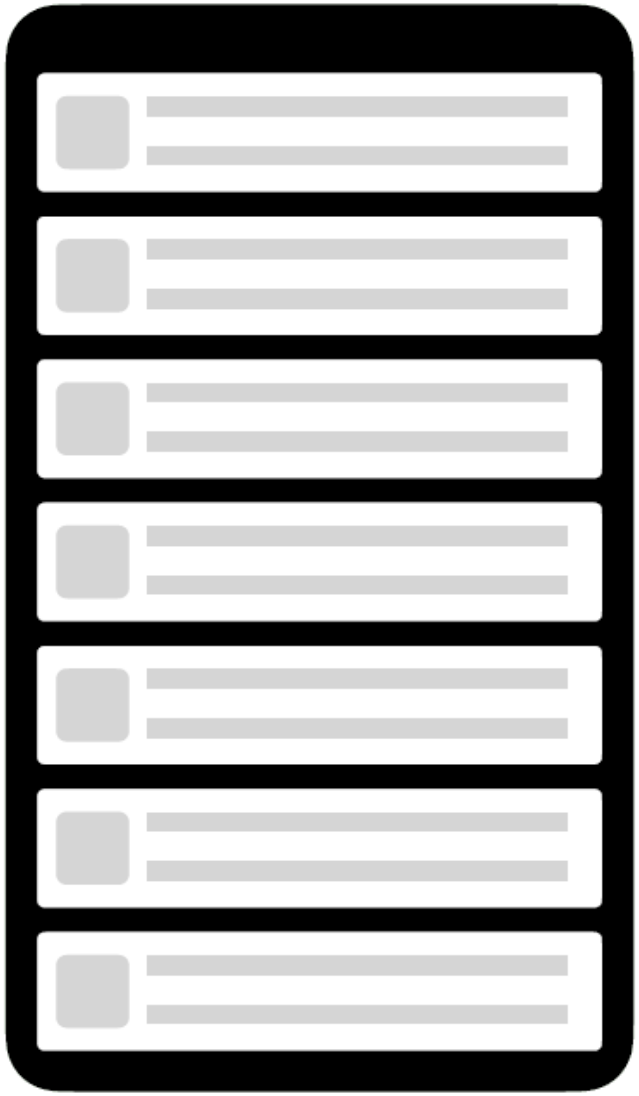
https://material.io/components

  - Float labels – TextInputLayout
  - FloatingActionButton
  - NavigationDrawer
  - Toolbar
  - CardView
  - TabLayout
  - BottomNavigationView
  - BottomSheet
  - Snackbar

AlertDialog

- TBD

List



```
@Composable
fun ListComponent(superheroList: List<Person>) {
    ScrollableColumn {
        for(person in superheroList) {
            SimpleRowComponent(
                person.name,
                person.age,
                person.profilePictureUrl
            )
        }
    }
}
```

List in Compose (with index)

```
@Composable
fun UserListScreen(users: List<User>) {
    LazyColumn(
        modifier = Modifier.fillMaxSize()) {
        item {
            Text("Header",
                Modifier.fillMaxWidth().padding(8.dp)
            )
        }
        itemsIndexed(users) { index, user ->
            Text("${user.name} - ${user.age}",
                Modifier.fillMaxWidth().padding(8.dp)
            )
        }
    }
}
```

Modifiers

Back

Modifiers

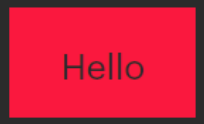
- It is like CSS for styling the app composables

  Used to provide **layout parameters** and **assign behavior**

- They're chained and the order matters!
  - They are applied in a sequential way and the order impacts the behavior

```
Text(
    text = "Hello",
    modifier = Modifier.padding(16.dp)
        .background(color = Color.Red)
)
```


```
Text(
    text = "Hello",
    modifier = Modifier.background(color = Color.Red)
        .padding(16.dp)
)
```



Photographer Card

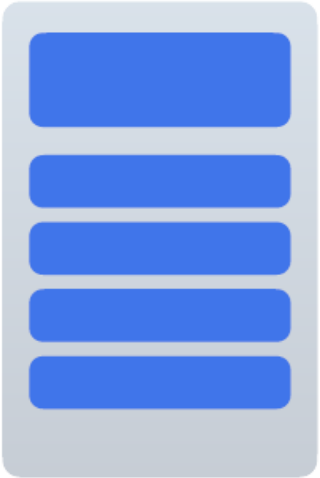
```
@Composable
fun PhotographerCard(
    photographer: Photographer,
    onClick: () -> Unit
) {
    val padding = 16.dp
    Column(
        modifier
            .clickable(onClick = onClick)
            .padding(padding)
            .fillMaxWidth()
    ) {
        Row(verticalGravity = Alignment.CenterVertically) { …
}
        Spacer(Modifier.size(padding))
        Card(elevation = 4.dp) { … }
    }
}
```

Layouts

Back

Compose Layout

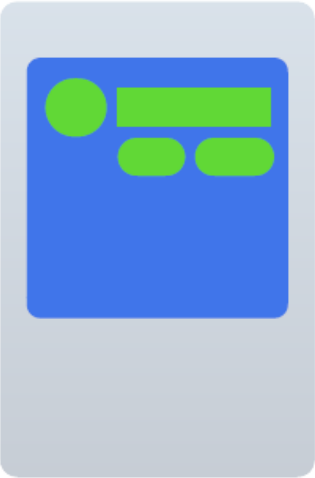
- Column = vertical orientation
- Row = horizontal orientation



Column

Row

Box



Constraint Layout

Row & Column Example

```
@Composable
fun ArtistCard(artist: Artist) {
    Row(verticalAlignment = Alignment.CenterVertically) {
        Image(/*...*/)
        Column {
            Text(artist.name)
            Text(artist.lastSeenOnline)
        }
    }
}
```

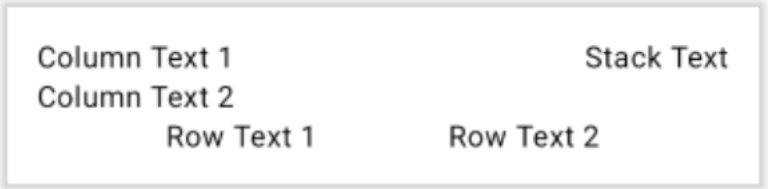
Alfred Sisley
3 minutes ago

Box Example

```
@Composable
fun ArtistAvatar(artist: Artist) {
    Box {
        Image(/*...*/)
        Icon(/*...*/)
    }
}
```

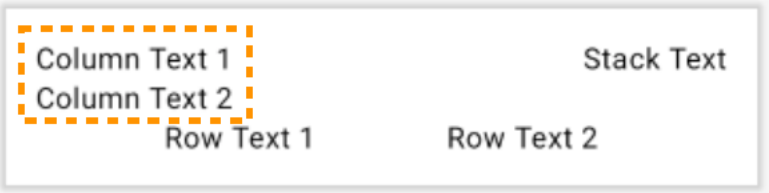
Box Example (1 of 4)

```
Box(modifier = Modifier.fillMaxWidth()) {
    Column(
        modifier = Modifier
            .padding(16.dp)
            .fillMaxWidth()
    ) {
        Text("Column Text 1")
        Text("Column Text 2")

        Row(
            modifier = Modifier.fillMaxWidth(),
            horizontalArrangement = Arrangement.SpaceEvenly
        ) {
            Text(text = "Row Text 1")
            Text(text = "Row Text 2")
        }
    }
    Text(
        "Stack Text",
        modifier = Modifier
            .align(Alignment.TopEnd)
            .padding(end = 16.dp, top = 16.dp)
    )
}
```



Box Example (2 of 4)

```
Box(modifier = Modifier.fillMaxWidth()) {
    Column(
        modifier = Modifier
            .padding(16.dp)
            .fillMaxWidth()
    ) {
        Text("Column Text 1")
        Text("Column Text 2")

        Row(
            modifier = Modifier.fillMaxWidth(),
            horizontalArrangement = Arrangement.SpaceEvenly
        ) {
            Text(text = "Row Text 1")
            Text(text = "Row Text 2")
        }
    }
    Text(
        "Stack Text",
        modifier = Modifier
            .align(Alignment.TopEnd)
            .padding(end = 16.dp, top = 16.dp)
    )
}
```

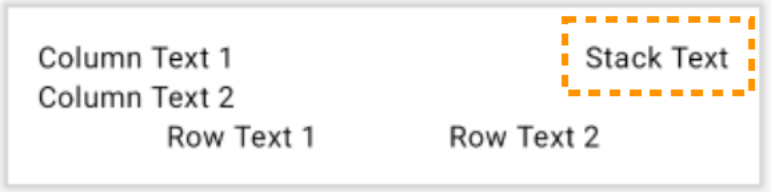


Box Example (3 of 4)

```
Box(modifier = Modifier.fillMaxWidth()) {
    Column(
        modifier = Modifier
            .padding(16.dp)
            .fillMaxWidth()
    ) {
        Text("Column Text 1")
        Text("Column Text 2")

        Row(
            modifier = Modifier.fillMaxWidth(),
            horizontalArrangement = Arrangement.SpaceEvenly
        ) {
            Text(text = "Row Text 1")
            Text(text = "Row Text 2")
        }
    }
    Text(
        "Stack Text",
        modifier = Modifier
            .align(Alignment.TopEnd)
            .padding(end = 16.dp, top = 16.dp)
    )
}
```

Box Example (4 of 4)

```
Box(modifier = Modifier.fillMaxWidth()) {
    Column(
        modifier = Modifier
            .padding(16.dp)
            .fillMaxWidth()
    ) {
        Text("Column Text 1")
        Text("Column Text 2")

        Row(
            modifier = Modifier.fillMaxWidth(),
            horizontalArrangement = Arrangement.SpaceEvenly
        ) {
            Text(text = "Row Text 1")
            Text(text = "Row Text 2")
        }
    }
    Text(
        "Stack Text",
        modifier = Modifier
            .align(Alignment.TopEnd)
            .padding(end = 16.dp, top = 16.dp)
    )
}
```



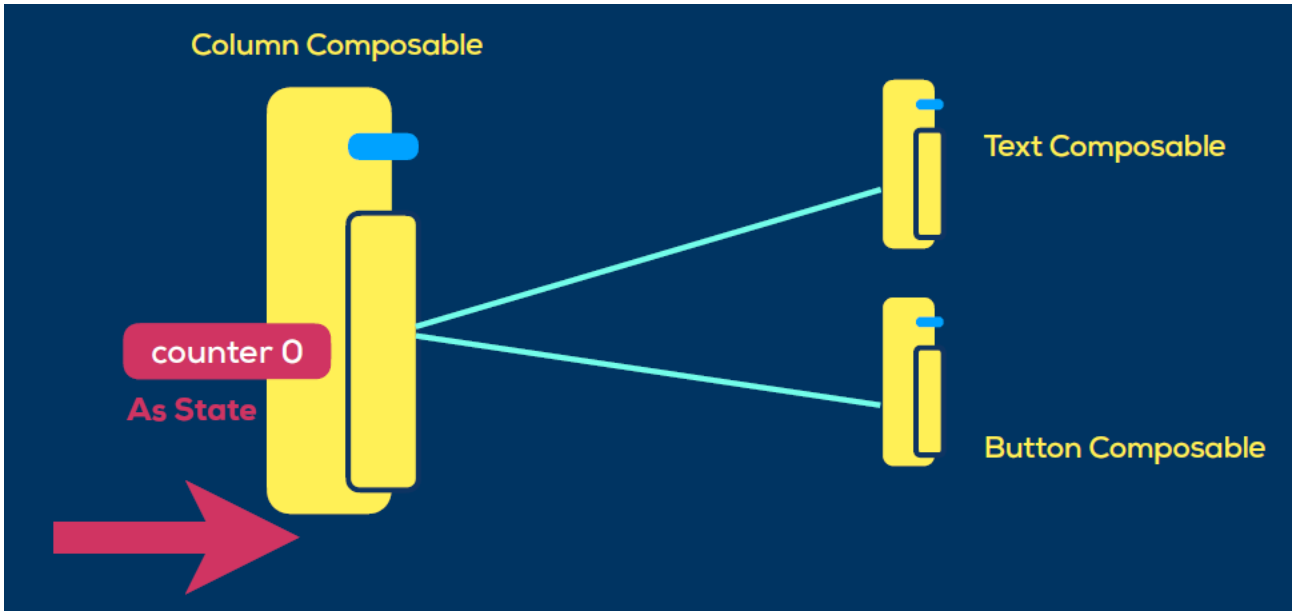
State

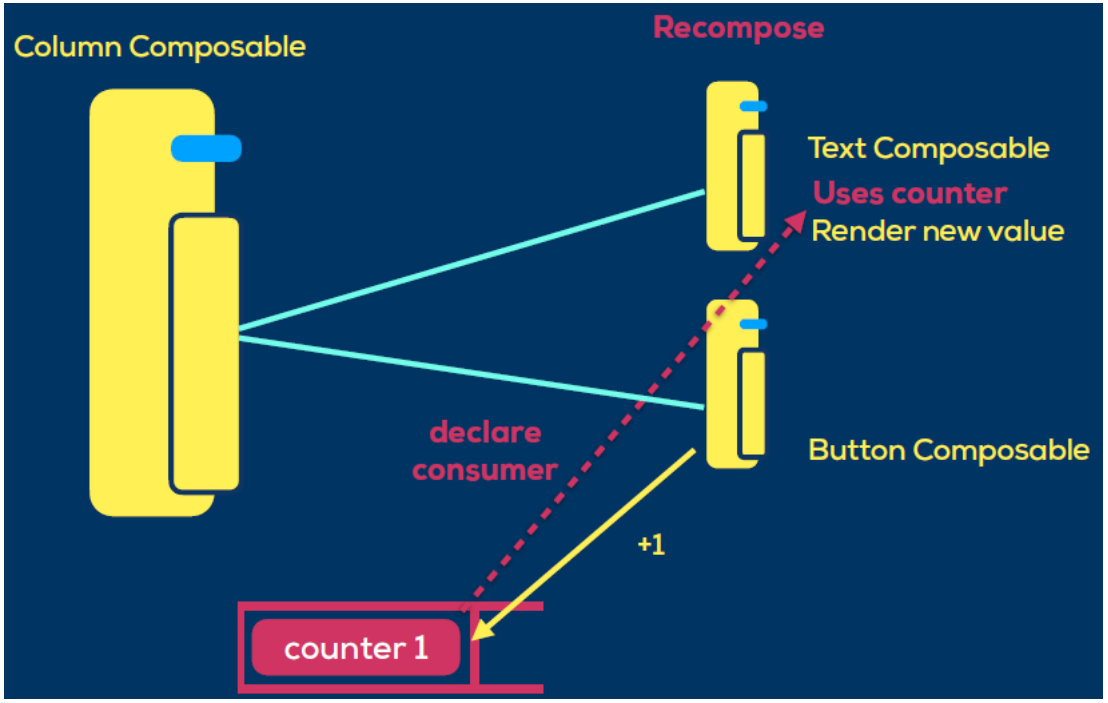
Back

State

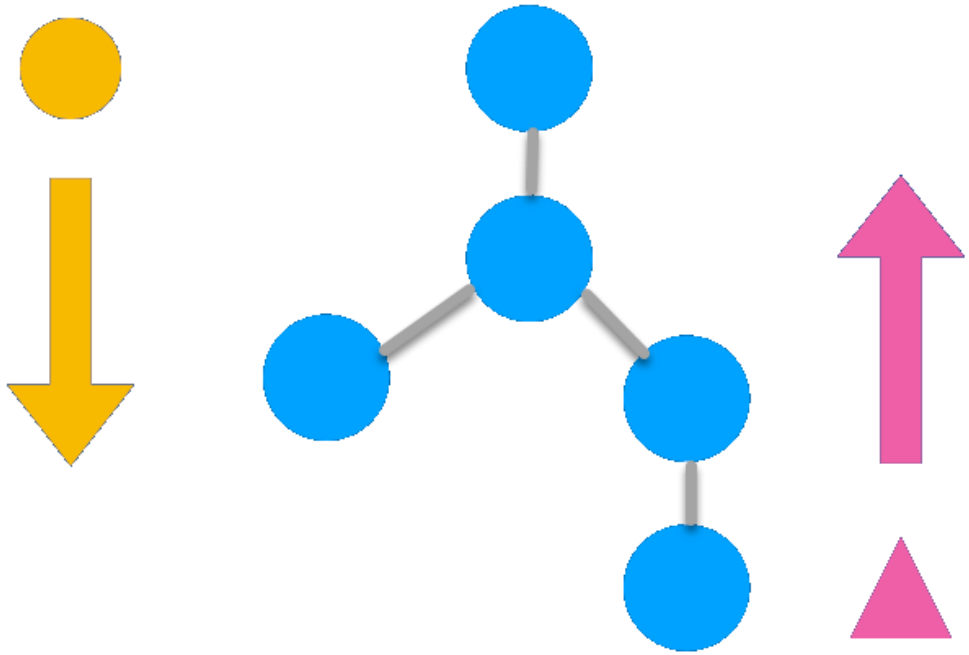
- State = value that can change overtime
- **Remember** in the composable memory to hold the state
- Any value changed in the state will recompose the composable
  - UI changes are handled by the Jetpack compose runtime not by the developer

```
var nameState by remember { mutableStateOf("") }
TextField(
    value = nameState,
    label = { Text("Name") },
    onValueChange = { s: String ->
        nameState = s
    }
)
```


Column Composable
Text Composable
counter 0
As State
Button Composable


Column Composable
Recompose
Text Composable
Uses counter
Render new value
declare
consumer
Button Composable
+1
counter 1

Unidirectional Data Flow



State flows down via parameters

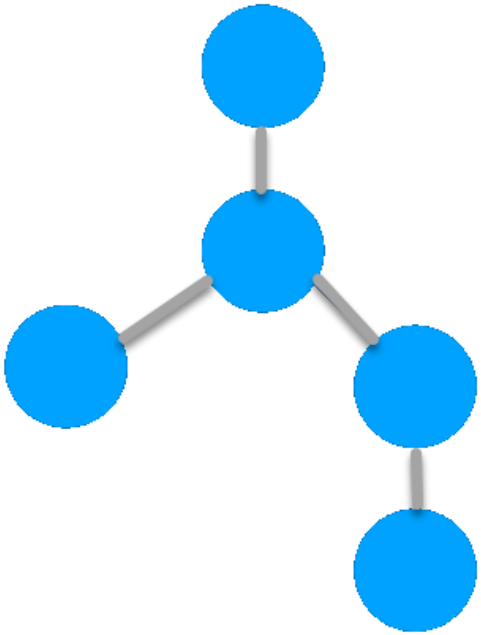
Events flow up via callbacks

Recomposition

- In an imperative UI model, to change a view, you call a setter on the view to change its internal state.
- In Compose, you call the composable function again with new data. Doing so causes the function to be recomposed--the view emitted by the function are redrawn, if necessary, with new data.
  - The Compose framework can intelligently recompose only the components that changed

How recomposition works

1. Creates an abstract representation of the UI and renders it
2. When a change occurs, it creates a new representation
3. Computes the differences between the two representations
4. Renders the differences [if any]



For more details about Jetpack Compose Runtime, watch this video

Summary

- **Activity** provides the UI that the user interacts with
  - It has layout (.xml) file & Activity class (UI Controller) => This allows a **clear separation** between the UI and the app logic
  - Activity class define listeners to handle events
- ConstraintLayout enables responsive design

.. mastering it will take some time and practice   …

Resources

- Jetpack compose tutorial

https://developer.android.com/jetpack/compose/tutorial

- Jetpack compose Code Labs

https://developer.android.com/courses/pathways/compose

- Compose Samples

https://github.com/android/compose-samples